COBB

Inside MICROSOFT ACCESS™

November 1993 • Vol. 1 No. 9

Tips & techniques for Microsoft Access • Windows

Query Tip

Using the DatePart( ) function in queries to group on date fields

As we've discussed many times in *Inside Microsoft Access*, queries are a very flexible tool for manipulating data. Even if the standard options don't adequately select or sort your data, you can use Access Basic functions such as DatePart( ) to do the job.

In this article, we'll describe how to use the DatePart( ) function to manipulate dates in queries. We'll also provide a general description of the function and work through some examples.

Understanding the DatePart( ) function

The DatePart( ) function accepts two arguments—the Date/Time value you want to analyze and a code that defines which component of the date you want to see. The function then returns that component. For instance, the function call

```
DatePrint("yyyy",[Order Date])
```

returns the year component of the current record's Order Date field entry.

Table A on page 2 lists the codes you can use in the DatePart( ) function and a brief description of each code. It also shows the value the function returns when you use each code to extract a part of the date November 1, 1993, at 9:00 A.M. As you can see, DatePart( ) also offers codes that extract the time part of the Date/Time value.

How DatePart( ) differs from Format( )

If you've experimented with Access Basic, you may be wondering how the DatePart( ) function differs from the Format( ) function, which can also return a component of the input date. Well, DatePart( ) always returns a numeric value; Format( ) always returns a text value.

Notice in Table A that the codes available to DatePart( ) are also codes Format( ) supports. These are the codes that return numeric information about the Date/Time value. For example, the code *d* tells the function to return the day of the month. If you use DatePart( ), it will return the number 1; if you use Format( ), the function will return the text value *1*.

Examples

We'll now look at two examples that demonstrate the DatePart( ) function's role in Access

IN THIS ISSUE

A Publication of The COBB Group

queries. We'll start with the easiest—using the function in a select query. Later, we'll show you how to arrange data in a crosstab query with the DatePart( ) function.

In both examples, we'll use the NWIND database's Orders table. Before beginning the examples, you should import the table into another database. That way, you can experiment with the data without damaging the table's data in the NWIND database.

After opening your test database, pull down the File menu and select the Import...

Table A

Code	Description	DatePart("<CODE>",<1-Nov 1993>)
yyyy	Returns the year value with the century digits. Possible values: 100 through 9999.	1993
q	Returns the quarter of the calendar year. Possible values: 1 through 4.	4
m	Returns the month as a number. Possible values: 1 through 12.	11
y	Returns the year without the century digits. Possible values: 1 through 366.	93
d	Returns the day as a number. Possible values: 1 through 31.	1
w	Returns the day of the week as a number, where 1 is Sunday, 2 is Monday, and so on. Possible values: 1 through 7.	2
ww	Returns the week of the year. Possible values: 1 to 53.	45
h	Returns the hour in military time. Possible values: 0 through 23.	9
n	Returns the minute. Possible values: 0 through 59.	0
s	Returns the second. Possible values: 0 through 59.	0

Inside MICROSOFT ACCESS™

Inside Microsoft Access (ISSN 1067-8204) is published monthly by The Cobb Group.

Prices:

Prices:	Domestic	$59/yr. ($7.00 each)
	Outside US	$79/yr. ($8.50 each)
Phone:	Toll free	(800) 223-8720
	Local	(502) 491-1900
	Customer Relations Fax	(502) 491-8050
	Editorial Department Fax	(502) 491-4200

Address:
You may address tips, special requests, and other correspondence to

The Editor, *Inside Microsoft Access*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

For subscriptions, fulfillment questions, and requests for bulk orders, address your letters to

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

Postmaster:
Second class postage is pending in Louisville, KY. Send address changes to

Inside Microsoft Access
P.O. Box 35160
Louisville, KY 40232

Authorized Canada Post International Publications Mail (Canadian Distribution) Sales Agreement #XXXXXX CANADA GST #123669673. Send returns to Canadian Direct Mailing Sys. Ltd., 920 Mercer Street, Windsor, Ontario, N9A 7C2. Printed in the USA.

Back Issues:
To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $7 each, $8.50 outside the US. You can pay with MasterCard, VISA, Discover, or American Express, or we can bill you. Please identify the issue you want by the month and year it was published. Customer Relations can also provide you with an issue-by-issue listing of all articles that have appeared in *Inside Microsoft Access*.

command. The dialog box that appears lists possible data formats; highlight *Microsoft Access* and click OK. In the next dialog box, named Select Microsoft Access Database, select the NWIND.MDB file from your \ACCESS directory and click OK. In the last dialog box, titled Import Objects, make sure the Object Type combo box shows Tables. Then, find the Orders table in the list and click the Import button. Access will display a dialog box telling you it successfully imported the table. Click OK in that dialog box and close the Import Objects dialog box.

Using DatePart( ) in a select query

There are two ways you can use DatePart( ) in select queries. You can use the function in the Field cell to generate the values for a new field, or you can use the function in the Criteria cell to define the records you want to select.

Figure A shows a query that uses the DatePart( ) function to fill a query field with the calendar year of each record's Order Date entry. To create the query, you add the fields Order ID and Customer ID by dragging them from the field list. Then, you enter in the third column's Field cell

```
Cal Year: DatePart("yyyy",[Order Date])
```

which names the new field Cal Year and then calls DatePart( ) to obtain the year component from the Order Date field entry.

Figure B shows a select query that uses DatePart( ) as criteria for a query. In this example, the query selects only records placed in the year 1992. To select the records, you drag the Order Date field as well as the Order ID and Customer ID fields from the field list. Then, in the Order Date column's Criteria cell, you enter the expression

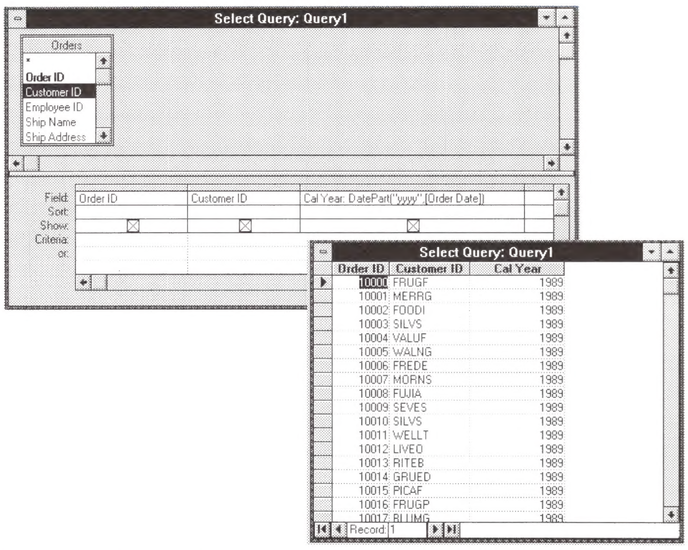
```
DatePart("yyyy",[Order Date]) = 1992
```

which evaluates to a True value only for records that store 1992 in the Order Date field.

Using DatePart( ) in a crosstab query

The DatePart( ) function is also very useful in creating crosstab queries. You use DatePart( ) to designate the ranges of Time/Date field entries that you want to group in the crosstab query's column headings. If you've never used crosstab queries, you may want to read

Figure A

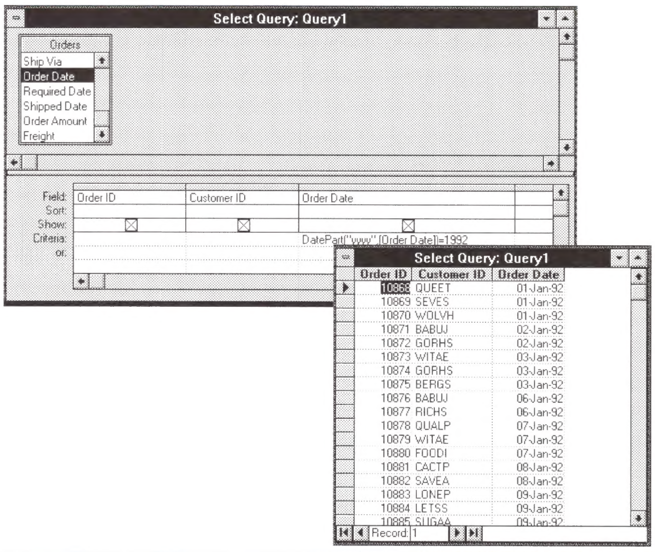


	Order ID	Customer ID	Cal Year: DatePart("yyyy",[Order Date])
Field:	Order ID	Customer ID	Cal Year: DatePart("yyyy",[Order Date])
Sort:			
Show:	☒	☒	☒
Criteria:			
or:			

Order ID	Customer ID	Cal Year
10000	FRUGF	1989
10001	MERRG	1989
10002	FOODI	1989
10003	SILVS	1989
10004	VALUF	1989
10005	WALNG	1989
10006	FREDE	1989
10007	MORNS	1989
10008	FUJIA	1989
10009	SEVES	1989
10010	SILVS	1989
10011	WELLT	1989
10012	LIVEO	1989
10013	RITEB	1989
10014	GRUED	1989
10015	PICAF	1989
10016	FRUGP	1989
10017	BLUMG	1989

This query creates a new field named Cal Year, which stores the calendar year of the Order Date entry.

Figure B



Field:	Order ID	Customer ID	Order Date
Sort:			
Show:	☒	☒	☒
Criteria:			DatePart("yyyy",[Order Date])=1992
or:			

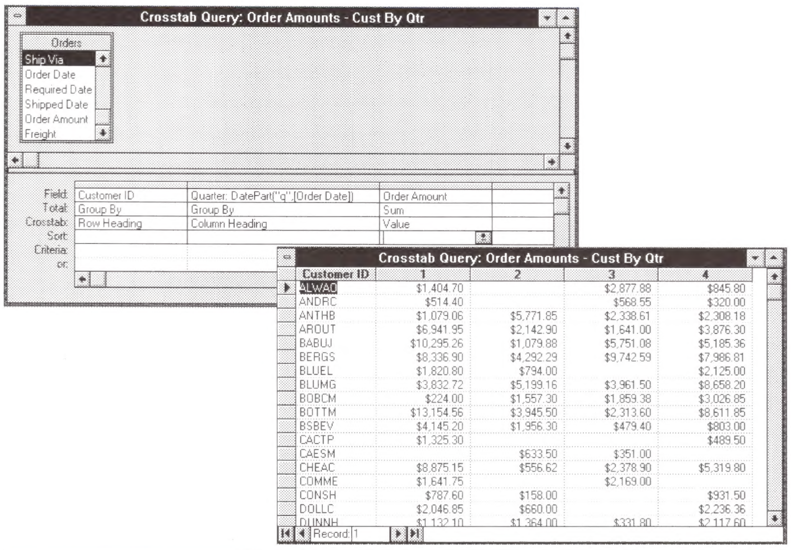
Order ID	Customer ID	Order Date
10868	QUEET	01-Jan-92
10869	SEVES	01-Jan-92
10870	WOLVH	01-Jan-92
10871	BABUJ	02-Jan-92
10872	GORHS	02-Jan-92
10873	WITAE	03-Jan-92
10874	GORHS	03-Jan-92
10875	BERGS	03-Jan-92
10876	BABUJ	06-Jan-92
10877	RICHS	06-Jan-92
10878	QUALP	07-Jan-92
10879	WITAE	07-Jan-92
10880	FOODI	07-Jan-92
10881	CACTP	08-Jan-92
10882	SAVEA	08-Jan-92
10883	LONEP	09-Jan-92
10884	LETSS	09-Jan-92
10885	SUGAA	09-Jan-92

By using the DatePart( ) function in the Criteria cell, you limit the records the query selects to those placed in 1992.

"Understanding Crosstab Queries," on page 6, before continuing with this example.

We'll create a crosstab query that totals the Orders table's Order Amount entries by customer and calendar quarters. The crosstab query will use the Customer ID field as its row-heading field. To create the column-heading field, the query will use an expression that calls the DatePart( ) function to return the calendar quarter of the Order Date field entries. The crosstab query will then group records by the calendar quarter of the Order Date entries.

Figure C


Crosstab Query: Order Amounts - Cust By Qtr

Field:	Customer ID	Quarter: DatePart("q",[Order Date])	Order Amount	
Total:	Group By	Group By	Sum	
Crosstab:	Row Heading	Column Heading	Value	
Sort:				
Criteria:				
or:				

Crosstab Query: Order Amounts - Cust By Qtr

Customer ID	1	2	3	4
ALWAO	$1,404.70		$2,877.88	$845.80
ANDRC	$514.40		$568.55	$320.00
ANTHB	$1,079.06	$5,771.85	$2,338.61	$2,308.18
AROUT	$6,941.95	$2,142.90	$1,641.00	$3,876.30
BABUJ	$10,295.26	$1,079.88	$5,751.08	$5,185.36
BERGS	$8,336.90	$4,292.29	$9,742.59	$7,986.81
BLUEL	$1,820.80	$794.00		$2,125.00
BLUMG	$3,832.72	$5,199.16	$3,961.50	$8,658.20
BOBCM	$224.00	$1,557.30	$1,859.38	$3,026.85
BOTTM	$13,154.56	$3,945.50	$2,313.60	$8,611.85
BSBEV	$4,145.20	$1,956.30	$479.40	$803.00
CACTP	$1,325.30			$489.50
CAESM		$633.50	$351.00	
CHEAC	$8,875.15	$556.62	$2,378.90	$5,319.80
COMME	$1,641.75		$2,169.00	
CONSH	$787.60	$158.00		$931.50
DOLLC	$2,046.85	$660.00		$2,236.36
DUNNH	$1,132.10	$1,364.00	$331.80	$2,117.60

This crosstab query sums quarterly Order Amount entries for each customer.

Start by highlighting the Orders table in the Database window and clicking the New Query button (⊞) on the toolbar. When the new Query window appears, pull down the Query menu and select Crosstab.

Next, add the fields you want to use as the row and column headings and the value field that the query actually totals. First, drag the Customer ID field from the Orders table's field list to the QBE grid. The entry *Group By* will appear in the Total cell. Move to the Crosstab cell, click the dropdown arrow, and choose *Row Heading* from the selection list.

Next, add the column-heading field. Move to the Field cell of the next column and type the expression

```
Quarters: DatePart("q",[Order Date])
```

This time, *Expression* is the Total cell's default value. Click the dropdown arrow and select *Group By* from the list. Next, move to the Crosstab cell, click that cell's dropdown arrow, and select *Column Heading* from the list.

Finally, you add the field you want the crosstab query to summarize—in this case, Order Amount. Return to the field list, scroll down until you find the Order Amount field, and drag that field to the third column of the QBE grid. Next, change the default *Group By* entry in the Total field to *Sum*. Then, in the Crosstab cell, click the dropdown arrow and select *Value* from the list.

Figure C shows the finished query and the summary results you see after clicking the

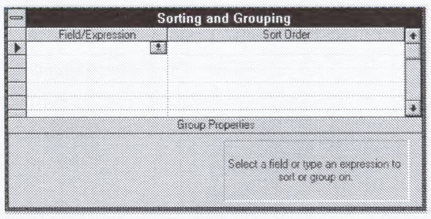
In a report, you don't need DatePart( ) to group by date components

You often want to group reports by date components. For instance, you may want to group orders by month or by quarter and then total the sales amounts in each group's footers.

You might think you'd need the function DatePart( ) to arrange the data properly, as we describe in the accompanying article. However, the Sorting and Grouping feature of Microsoft Access reports lets you group on date components without having to use any special functions.

As you may know, you specify how reports sort and group data in the Sorting and Grouping window shown in Figure A. Let's briefly review how to use this window to group your data. Then, we'll show you how this feature lets you group by date components.

Figure A



Select a field or type an expression to sort or group on.

You define how a report sorts and groups data by using the Sorting and Grouping window.

The Sorting and Grouping window

The Sorting and Grouping window contains a grid in which you enter the fields and expressions you want the report to group on. In the Field/Expression column, you enter the field or expression Access will use to group the report. For example, in Figure B, the first row's Field/Expression cell contains the field Order Date. Access will sort this report by the Order Date field entries.

Datasheet button (▦) on the toolbar. Before continuing, save the query by pulling down the File menu, selecting Save Query As…, and entering *Order Amounts - Cust By Qtr*.

A common DatePart( ) trap

Actually, the summary results shown in Figure C are a little misleading. If you've worked with the NWIND database before, you might know that the Orders table stores several years' worth of order information. As a result, the query mixes the order amounts from all years when it computes the quarterly totals. For instance, the first quarter's column, labeled *1*, shows the combined total of each customer's first-quarter Order Amount for every year stored in the database. If you're looking for trends in your customers' purchasing habits, this may be what you want. However, if you need to see *each year's* quarterly sales figures, you'll have to modify the crosstab query.

This is a common trap you'll encounter when you use DatePart( ) to create a *Group By* field in any query that computes totals. DatePart( ) simply returns a number that the query will group the data on. The number itself doesn't communicate anything about the original date. Therefore, you must ensure the query groups the date components in an appropriate manner.

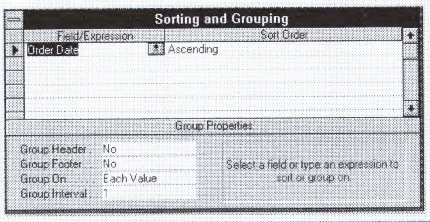
In this example, you want to group calendar quarters within calendar years, so you must first tell the query to group on the year component of the date and then group on the quarter. The easiest way to accomplish this grouping is to add a second row-heading field that groups by the year of the Order Date field entries.

You can start with the Order Amounts - Cust By Qtr query. If you left the query onscreen, just click the Design View button (📐) on the toolbar. Next, highlight the Customer ID field's column selector and press [Ins]. A new column will appear in the first column position. In this column, enter in the Field cell

```
Year: DatePart("yyyy",[Order Date])
```

This expression names the new query field Year and then generates the year component of the Order Date field. Then, enter *Group By* in the Total cell and *Row Heading* in the Crosstab cell.

Figure B



Sorting and Grouping

Field/Expression	Sort Order
Order Date	Ascending

Group Properties

Group Header . . No
Group Footer . . No
Group On . . . . . Each Value
Group Interval . . 1

Select a field or type an expression to sort or group on.

When you enter Order Date *in the Field/Expression cell, the report will sort on the Order Date field entries.*

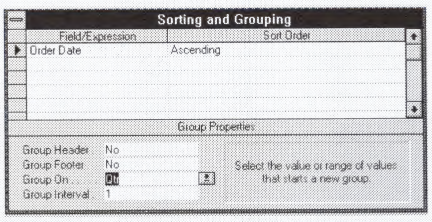
The Sorting and Grouping window also contains a Group Properties section below the grid, in which you specify exactly how you want the report to group the data. You define how you want the report to cluster the data by setting the properties Group Header, Group Footer, Group On, and Group Interval. In this article, we're concerned with the Group On property, which defines the type of value range the group will cluster.

Grouping records on a Date/Time field

To group on a date component, you enter the Date/Time field you want to group on. Then, you move to the Group On property, click on the dropdown arrow, and select the date component on which you want to group the data. The selection list offers the choices *Each Value*, *Year*, *Qtr*, *Month*, *Week*, *Day*, *Hour*, and *Minute*.

Returning to the example, suppose you're sorting data by the Order Date field. You then decide to group by the calendar quarter. Just select *Qtr* from the Group On property's selection list, as shown in Figure C.

Figure C



Sorting and Grouping

Field/Expression	Sort Order
Order Date	Ascending

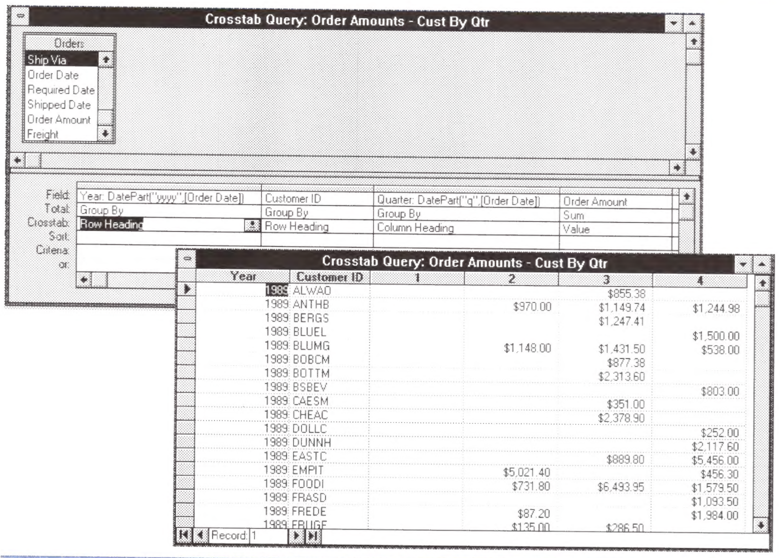
Group Properties

Group Header . . No
Group Footer . . No
Group On . . . . . Qtr
Group Interval . . 1

Select the value or range of values that starts a new group.

To group data by calendar quarters, set the Group On property to Qtr.

Figure D



Field:	Year: DatePart("yyyy",[Order Date])	Customer ID	Quarter: DatePart("q",[Order Date])	Order Amount
Total:	Group By	Group By	Group By	Sum
Crosstab:	Row Heading	Row Heading	Column Heading	Value
Sort:				
Criteria:				
or:				

Crosstab Query: Order Amounts - Cust By Qtr

Year	Customer ID	1	2	3	4
1989	ALWAO			$855.38	
1989	ANTHB		$970.00	$1,149.74	$1,244.98
1989	BERGS			$1,247.41	
1989	BLUEL				$1,500.00
1989	BLUMG		$1,148.00	$1,431.50	$538.00
1989	BOBCM			$877.38	
1989	BOTTM			$2,313.60	
1989	BSBEV				$803.00
1989	CAESM			$351.00	
1989	CHEAC			$2,378.90	
1989	DOLLC				$252.00
1989	DUNNH				$2,117.60
1989	EASTC			$889.80	$5,456.00
1989	EMPIT		$5,021.40		$456.30
1989	FOODI		$731.80	$6,493.95	$1,579.50
1989	FRASD				$1,093.50
1989	FREDE		$87.20		$1,984.00
1989	FRUGE		$135.00	$286.50	

To separate the many years of quarterly order amount totals, you must use a second row-heading field.

Figure D shows the resulting query and its datasheet. As you can see, the first field now displays the year of the order. As a result, the summary totals in columns 1, 2, 3, and 4 display quarterly order amounts for each year. Before closing the query, pull down the File menu and select the Save Query As... command. Then, type *Order Amounts - Year/Cust By Qtr* and click OK.

Conclusion

In this article, we've introduced you to the Access Basic function DatePart( ), which returns a component of a date value. You can use this function to generate values for the query field's entries. You can also use the function in the Criteria cell to limit the records by date component.

The DatePart( ) function is also very useful when you're computing summary totals. You can use the function to group data over ranges of dates. The example we showed you uses DatePart( ) to create column-heading fields in a crosstab query that displays quarterly totals. ◆

Query Tip

Understanding crosstab queries

If you've never used crosstab queries, you may be confused about when to use them for computing summary totals. You may also not understand how to choose between a crosstab query and an ordinary select query for summarizing your data. In this article, we'll describe the advantages of using a crosstab query and compare the way the two types of queries arrange the summary results. We'll also show you how to convert a select query to a crosstab query.

As you may know, you use a crosstab query when you want to summarize data after grouping the data by two or more fields. For instance, in "Using the DatePart( ) Function in Queries to Group on Date Fields," on page 1, we created a crosstab query that computed quarterly sales figures for each customer in the NWIND database. Specifically, we grouped Order Amount field entries by the Customer ID field and the calendar quarter of the Order Date field.

Let's start by using an ordinary select query to compute these summary totals. To do so, you first include the fields by which you want to group the data and enter *Group By* in their Total cells. You then include the field you want to summarize and enter the appropriate summary operator in its Total cell. Highlight the Orders table in the Database window and click the New Query button (▤) on the toolbar. When the new Query window appears, drag the Customer ID, Order Date, and Order Amount fields to the QBE grid. Then, modify the *Order Date* entry in the second column's Field cell so that it reads

```
Quarter: DatePart("q",[Order Date])
```

This expression names the field Quarter and generates the calendar quarter of the Order Date entries rather than the full date.

Next, click the Totals button (Σ) on the toolbar. When you do, Access opens a new row named Total in the QBE grid. The new row will display the entries *Group By*, *Expression*, and *Group By* (respectively) in the three fields' cells. To compute the summary totals you want, you must change the second column's Total cell entry to *Group By* and the third column's Total cell entry to *Sum*. Figure A shows the completed query and the datasheet the query generates.

As you can see, the query generates a separate row for each unique combination of Customer ID entry and calendar quarter. Of course, the summary results are correct. However, you can't easily compare the different customers' quarterly totals. It would be nice to break out each quarter's summary information into separate columns. As a result, you'd see all first-quarter totals in one column, all second-quarter totals in another, and so on.

Well, crosstab queries arrange the summary results in just this way. You can select one of the grouping fields to be a column-heading field so that the results of each group appear in separate columns.

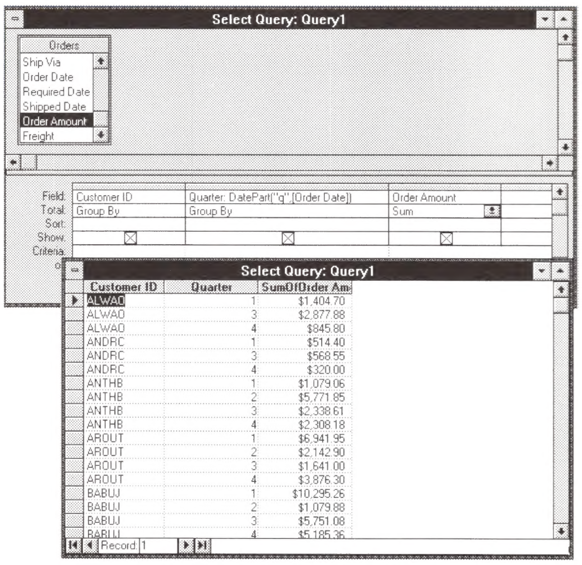
In our example, you can create a crosstab query that displays in separate columns the order amount totals for the customers' four quarterly totals. The datasheet's rows will list the customers.

You can easily create such a query by adapting the select query you created earlier. Just click on the Design View button (📐) on the toolbar, pull down the Query menu, and select Crosstab. When you do, a new row, named Crosstab, will open in the QBE grid under the Total row. In the Crosstab cells, you specify the row-heading field (or fields), the column-heading field, and the value field.

In this example, you enter *Row Heading* in the Customer ID column's Crosstab cell, *Column Heading* in the Quarter column's cell, and *Value* in the Order Amount column's cell. Note that the Crosstab cells offer a dropdown arrow and selection list that help you enter these settings. Figure B shows the resulting query and its datasheet.

Many people compare crosstab datasheets to spreadsheets. If you use spreadsheets, you'll probably see the analogy immediately. Spreadsheets provide a very flexible workspace in which you can arrange data in a visually appropriate way. Our example is a good illustration. If you were using a spreadsheet to compile quarterly sales totals, you'd naturally use separate columns for each quarter. Databases such as Access store data in a much more rigid format. You must therefore use a special tool, such as a crosstab query, to arrange the data in just the right way.

Figure A



Field:	Customer ID	Quarter: DatePart("q",[Order Date])	Order Amount
Total:	Group By	Group By	Sum
Sort:			
Show:	☒	☒	☒
Criteria:			

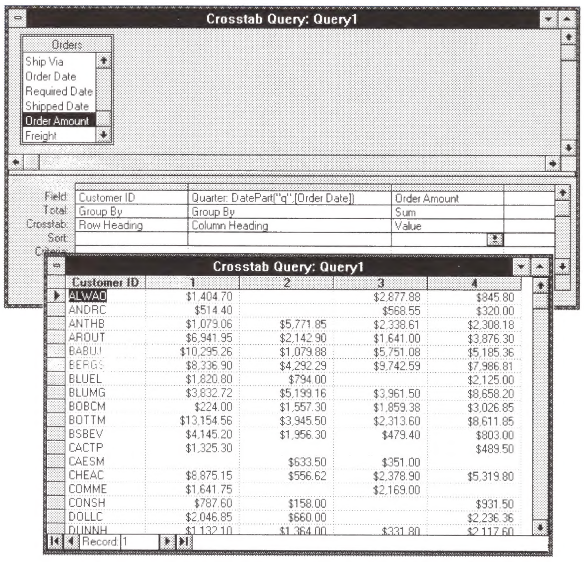
Select Query: Query1

Customer ID	Quarter	SumOfOrder Am
ALWAO	1	$1,404.70
ALWAO	3	$2,877.88
ALWAO	4	$845.80
ANDRC	1	$514.40
ANDRC	3	$568.55
ANDRC	4	$320.00
ANTHB	1	$1,079.06
ANTHB	2	$5,771.85
ANTHB	3	$2,338.61
ANTHB	4	$2,308.18
AROUT	1	$6,941.95
AROUT	2	$2,142.90
AROUT	3	$1,641.00
AROUT	4	$3,876.30
BABUJ	1	$10,295.26
BABUJ	2	$1,079.88
BABUJ	3	$5,751.08
BABUJ	4	$5,185.36

Record: 1

A select query computes summary totals as shown here.

Figure B



Field:	Customer ID	Quarter: DatePart("q",[Order Date])	Order Amount
Total:	Group By	Group By	Sum
Crosstab:	Row Heading	Column Heading	Value
Sort:			

Crosstab Query: Query1

Customer ID	1	2	3	4
ALWAO	$1,404.70		$2,877.88	$845.80
ANDRC	$514.40		$568.55	$320.00
ANTHB	$1,079.06	$5,771.85	$2,338.61	$2,308.18
AROUT	$6,941.95	$2,142.90	$1,641.00	$3,876.30
BABUJ	$10,295.26	$1,079.88	$5,751.08	$5,185.36
BERGS	$8,336.90	$4,292.29	$9,742.59	$7,986.81
BLUEL	$1,820.80	$794.00		$2,125.00
BLUMG	$3,832.72	$5,199.16	$3,961.50	$8,658.20
BOBCM	$224.00	$1,557.30	$1,859.38	$3,026.85
BOTTM	$13,154.56	$3,945.50	$2,313.60	$8,611.85
BSBEV	$4,145.20	$1,956.30	$479.40	$803.00
CACTP	$1,325.30			$489.50
CAESM		$633.50	$351.00	
CHEAC	$8,875.15	$556.62	$2,378.90	$5,319.80
COMME	$1,641.75		$2,169.00	
CONSH	$787.60	$158.00		$931.50
DOLLC	$2,046.85	$660.00		$2,236.36
DUNNH	$1,132.10	$1,364.00	$331.80	$2,117.60

Record: 1

The crosstab query we show here breaks the summary totals for the quarters into separate columns.

Conclusion

In this article, we described how crosstab queries can arrange summary results more quickly than a select query. We then demonstrated the difference between crosstab and select queries with an example. ◆

Form Tip

Allowing blank values in combo boxes that limit entries to the list

As you probably know, combo box controls offer a list of values you can choose from as you enter data. One of the nicer aspects of combo boxes is their ability to restrict entries to only those values in the list. To activate this restriction, you simply set the Limit To List property to *Yes* while designing the form.

However, the Limit To List property has one characteristic that can be a problem: Once you've entered a value in the combo box, you can't change your mind and leave a blank value in the combo box. Access will force you to select an entry from the list.

In this article, we'll show you how to set up a validation rule for the combo box control that emulates the Limit To List property but, at the same time, permits blank values. As you'll see, setting up a validation rule is easy, but some of the concepts behind validation rules aren't. We'll first explain the general principles of validation rules.

What is a validation rule?

A validation rule is an expression that defines valid entries for a field or form control. The form of the expression that validates your entry depends on the situation. However, the underlying characteristic of all validation-rule expressions is they always evaluate to a True or False value. A True value approves the entry, and a False value rejects it.

Let's look at a simple example. One common type of validation rule limits entries to a value higher than some lower boundary. For instance, suppose you want the values you enter in a text box called Number to be greater than 100. To set up a validation rule, you open the property sheet and, in the control's Validation Rule property, enter the expression

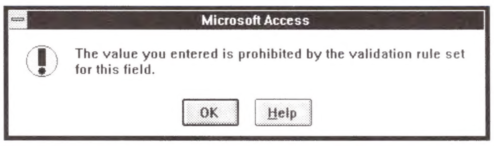
```
[Number] > 100
```

or simply

```
> 100
```

After you do, you'll be able to enter only values above 100 in the Number text box. If you try to enter *50*, Access will display the dialog box shown in Figure A.

Figure A


Microsoft Access

The value you entered is prohibited by the validation rule set for this field.

If you enter a value that fails the control's validation rule, Access displays this dialog box.

The limit-to-list validation rule that permits blank entries

Now that we've discussed how validation rules let you restrict field entries, it's time to discuss the validation rule that permits blank values in combo boxes. Our validation rule contains two expressions connected by the Or keyword. One expression will check whether the table or query that defines the list contains the combo box entry, and the other will check for a blank entry. By combining these expressions with the Or keyword, the validation rule expression will evaluate to True when either expression evaluates to a True value.

As you'll see, the expressions that the validation rule comprises can be complex. Therefore, we'll discuss them in terms of a specific example. Let's assume you're designing a form called Order Entry. The form contains a combo box named Cust ID Combo that provides the Cust ID entries of the Customers table in the combo box's selection list. You've bound the combo box to the field Cust ID of the form's main table, Invoices. The validation rule you'll create makes sure that your entries into the combo box are either blank or already in the Customers table.

We'll start with the expression that checks for blank entries, since it's the easier of the two. This particular expression must simply test whether the control value equals an empty string. For example, the expression

```
[Cust ID Combo]= ""
```

tests for a blank entry in the control named Cust ID Combo.

Now, the other piece of the validation rule's expression verifies that the entry is in the list. To do so, it uses the DLookup( ) function. If you're unfamiliar with DLookup( ), you may want to read the sidebar "Understanding DLookup( ) and the Other Domain Functions," on page 10, for some background information.

You want the function to look up the field value of the combo box's bound field and return it *only* if your entry in the combo box control resides in the table. The function call

```
DLookup("[Cust ID]", "Customers",
➥      "[Cust ID] =
➥      Forms![Order Entry]![Cust ID Combo]")
```

looks in the Customers table's Cust ID field for a record having a Cust ID entry that matches the Cust ID entry you've entered in the Cust ID Combo combo box control. If the combo box entry matches an item in the list, it simply returns the same value. If it doesn't match an item, the function returns a Null value. Remember, the ➥ character signifies that the line continues from the previous one.

In this situation, you aren't interested in the actual value the function returns. The important thing to remember is that the function will return either a field value or a Null value. You create the validation rule to reject entries for which the function call returns a Null value, since a Null value means that the entry doesn't have a matching value in the list. The expression you actually use in the validation rule for this example is

```
DLookup("[Cust ID]", "Customers",
➥   "[Cust ID] =
➥   Forms![Order Entry]![Cust ID Combo]")
➥   Is Not Null
```

Now you're ready to put the expressions together to form the full validation-rule expression for the Cust ID Combo combo box. Once you place the control on the form, open the property sheet and assign to the validation rule property the long expression

```
[Cust ID Combo]= "" Or
➥   DLookup("[Cust ID]", "Customers",
➥   "[Cust ID] =
➥   Forms![Order Entry]![Cust ID Combo]")
➥   Is Not Null
```

An example

Let's build a simple form that uses this validation rule. Tables A and B show the structures of the Customers and Invoices tables, respectively. Figure B shows some sample data for the Customers table. (You don't need sample data for the Invoices table, since you'll be adding data to that table.)

Now let's create the form. Start by highlighting the Invoices table's name in the Database window and clicking the New Form button (▤) on the toolbar. In the dialog box that appears, click the FormWizards button. Then, highlight the Single-column entry in the next dialog box and click OK. When the form wizard's first dialog box appears, just click the Fast Forward button (>>|) in order to generate the default form. In the last dialog box, replace the default title of the form with *Order Entry*. Then, click the Design button. Access will create the basic form shown in Figure C.

Table A

Customers Table

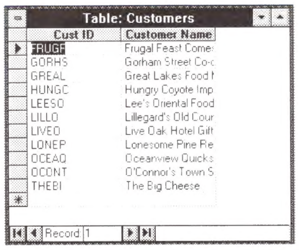
Key	Field Name	Data Type	Field Properties
🔑	Cust ID	Text	Field Size = 5
	Customer Name	Text	Field Size = 40

Table B

Invoices Table

Key	Field Name	Data Type	Field Properties
🔑	Invoice ID	Counter	
	Invoice Date	Date/Time	
	Cust ID	Text	Field Size = 5
	Amount	Currency	

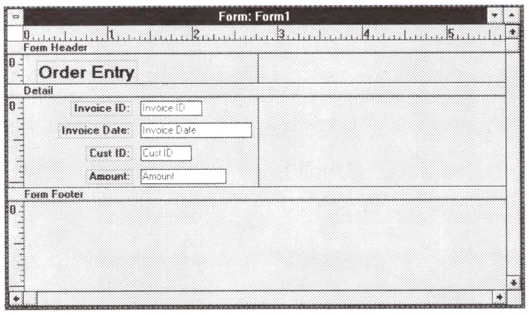
Figure B



Cust ID	Customer Name
FRUGF	Frugal Feast Come:
GORHS	Gorham Street Co-c
GREAL	Great Lakes Food I
HUNGC	Hungry Coyote Imp
LEESO	Lee's Oriental Food
LILLO	Lillegard's Old Cour
LIVEO	Live Oak Hotel Gift
LONEP	Lonesome Pine Re
OCEAQ	Oceanview Quicks
OCONT	O'Connor's Town S
THEBI	The Big Cheese

We'll use this data for the Customers table in our example.

Figure C



We used the Single-column form wizard to generate this simple form for our example.

Next, you replace the Cust ID text box with a combo box. First, delete the text box by highlighting it and then pressing [Del]. Then, place the combo box. To do so, you need to make sure both the toolbox and the field list are available on the desktop. Then, you select the Combo Box tool (▤) in the toolbox, click on the Cust ID field name in the field list, and drag the mouse pointer to the spot in which you want to place the combo box. When you release the mouse, the new combo box will appear. Next, reset the form's tab order by issuing the Edit menu's Tab Order command. When the Tab Order dialog box appears, click the Auto Order button and then click OK.

Next, you click the Properties button (▤) on the toolbar and assign the properties that define the combo box's operation. First, assign *Cust ID Combo* to the Control Name property. Then, move to the Row Source

Understanding DLookup( ) and the other domain functions

In the accompanying article, we used the DLookup( ) function to determine whether a value you enter in a combo box resides in a table or query. In this short article, we'll describe in detail how you use this function.

The DLookup( ) function searches for a value in a table or query without your having to open the table or query in a window. Furthermore, it's very fast compared to a query that selects the value in a record set.

The function accepts three arguments that provide the information it needs to look up a particular value. The first argument accepts the field that holds the data you want to look up; the second argument accepts the name of the table or query in which the data resides; the third argument accepts a conditional expression that specifies the record that contains the data you want.

Let's look at an example. Suppose you have a table named Invoice that stores customer orders. The function call

```
DLookup("[Order Amount]",
➡      "Invoice",
➡      "[Order Number] = 10001")
```

returns the purchase amount of the order in the Invoice table having the Order Number entry *10001*.

Other domain functions

The *D* in DLookup stands for *domain*, which is any record set. The most common domains are tables and queries. However, snapshots are also domains. (A snapshot is exactly what it sounds like—a picture of a table's or query's data at a certain point in time. You can create snapshots only with Access Basic functions and methods.)

DLookup( ) is one of the domain functions that Access Basic provides for determining information about a record set. Although these functions do different things, each retrieves some sort of information about the data in a domain. And like DLookup( ), all the domain functions are much faster than functionally equivalent queries. For instance, DMax( ) returns the largest value in a certain field of a table. Table A lists the domain functions Access Basic provides, along with brief explanations.

Table A

Access Basic's Domain Functions

Function	Description
DAvg( )	Returns the average (or, more precisely, the arithmetic mean) of a set of values in a record set
DCount( )	Returns the number of values in a domain
DFirst( )	Returns a field value from the first record in a record set
DLast( )	Returns a field value from the last record in a record set
DLookup( )	Returns a field value in a record set
DMin( )	Returns the minimum value in a record set
DMax( )	Returns the maximum value in a record set
DSDev( )	Returns the standard deviation of a population sample that's defined by a record set
DSDevP( )	Returns the standard deviation of a population that's defined by a record set
DSum( )	Returns the sum of a field's values in a record set
DVar( )	Returns the variance of a population sample that's defined by a record set
DVarP( )	Returns the variance of a population that's defined by a record set

property, click its dropdown arrow, and select Customers from the list of tables and queries.

The only other property you must set is the Validation Rule. We're using the same control and field names as before, so you simply enter the validation-rule expression from that example:

```
[Cust ID Combo]= "" Or
➥ DLookup("[Cust ID]", "Customers",
➥ "[Cust ID] =
➥ Forms![Order Entry]![Cust ID Combo]")
➥ Is Not Null
```

Using the form

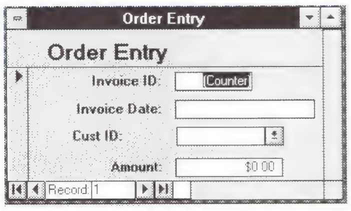
Before you can view the form, pull down the File menu and select the Save As... command. Enter *Order Entry* and click OK. Note that you must name this form Order Entry because you referred to it by that name in the DLookup( ) function's third argument, which provides the conditional expression

```
[Cust ID] = Forms![Order Entry]![Cust ID
    Combo]")
```

Once you've named the form, click the Form View button (▤) on the toolbar and try to enter a few values in the combo box. Figure D shows the finished form.

You can enter items that are in the list or you can leave the control blank. However, you can't type values that aren't in the list. For example, if you type *00000* into the combo box and then try to leave the field, Access will produce the dialog box we showed you in Figure A.

Figure D



Your completed form lets you select a Cust ID entry by using a combo box.

Notes

Access' refusal to accept blank values when you activate the Limit To List property can often be a benefit rather than a problem. For instance, in many situations, you'd always want to enter a Cust ID value on the Order Entry form. In fact, if you ever leave this combo box blank, you could be asking for trouble because you'd store an order without a customer. In those cases in which you want the combo box to permit blank values, you can't rely on the Limit To List property to set up the combo box the way you want.

Conclusion

In this article, we showed you how to create a validation rule that restricts entries to the list values and blank values. Although we described the technique with a specific example, you can use this type of validation rule for any combo box. ◆

Using SHARE to prevent simultaneous access to your database files

Access Tip

Beginning with Version 3.0, Windows provides multitasking capability in 386 Enhanced mode, which means you can run multiple applications at the same time. However, during multitasking operations, more than one application—or separate instances of the same application—can simultaneously access files on your hard drive. If this occurs, the file might not contain the data you expect, or it might even become corrupted and therefore useless.

Access is susceptible to this problem—just as any application is. If you open the same database with different instances of Access, you could seriously damage the database file.

Fortunately, you avoid the problem by loading the DOS SHARE program before launching Windows. SHARE provides file-locking capabilities similar to the file locking you'd find on a network such as Novell's NetWare. SHARE can sense when other applications are using a file and prevents simultaneous access to the file by applications that aren't designed to share the file.

In this article, we'll explain the protection SHARE provides your files, and we'll demonstrate why you must use SHARE when running Microsoft Access in Windows 3.0 and 3.1. Note that we haven't mentioned Windows for Workgroups. As you'll see, Windows for Workgroups is a different story.

What can happen without SHARE running?

If you're new to the Windows environment, you might not think you'd ever encounter a situation in which you'd access a file you already have open. As you begin to exploit the benefits of multitasking in Windows, you'll probably find that you typically have several applications running at the same time. For instance, you may open Microsoft Word while running Access.

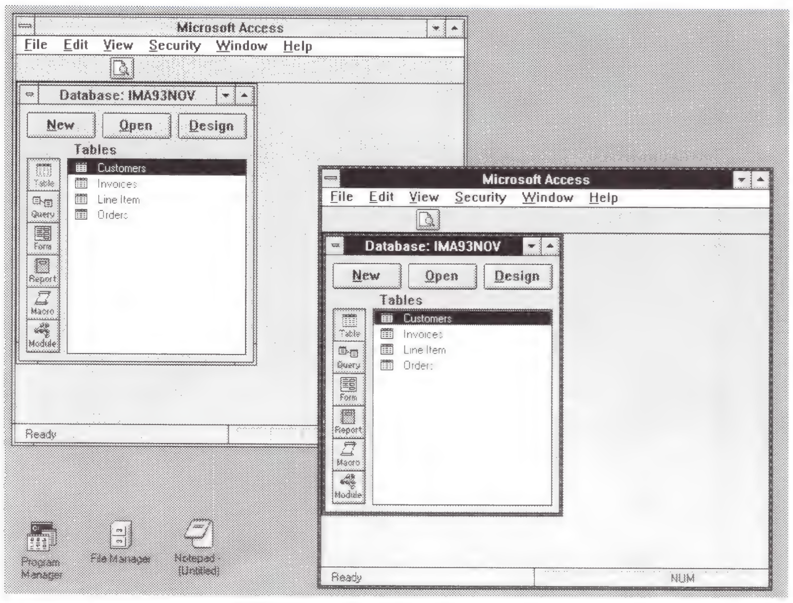
As you'd expect, when you run several applications at a time, you may forget which ones are open. In fact, you may inadvertently launch the same application twice! For example, suppose you run Access and open a database. While you're editing that file, you switch to other applications to perform various tasks. After completing those tasks, you might forget you've already opened the database and launch another copy—or instance—of Access and reopen the same database. Figure A shows how we launched two instances of Access to work with the same database file. Of course, to create this figure, we started Windows without first running the SHARE program.

Without SHARE running, Windows will let you open the same database file twice. However, doing so can have disastrous results because each instance of Access edits its own unique copy of the database file. Consequently, changes you make in one instance of the program are *not* reflected in the other.

Let's look at an example of the problems that can arise. Suppose you could enter Design view and modify the fields of a table in one instance of Access while editing the data in that table in another instance. If you were allowed to do so, you could remove a field from the table that you were simultaneously adding data to.

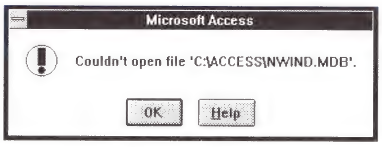
As we mentioned, you can prevent this type of problem by loading SHARE before launching Windows. SHARE prevents applications from accessing the same file in the Windows environment. That way, when you try to open a database file with a second instance of Access, Access will detect the potential conflict and display the dialog box shown in Figure B. As you can see, this dialog box doesn't offer choices—it simply tells you that you can't open the file.

Figure A



If you don't run SHARE before starting Windows, you can open the same database in separate instances of Access.

Figure B



Couldn't open file 'C:\ACCESS\NWIND.MDB'.

Access pops up this dialog box when you try to open a database that you've already opened with another copy of Access.

Is your computer running SHARE?

If you installed Access with all the default options, your computer is already running SHARE. You may remember that at the end of the installation process, Access' SETUP program asked you whether it could make changes to your AUTOEXEC.BAT file. As long as you gave your permission—the default response—SETUP inserted in the first line of your AUTOEXEC.BAT file the statement

```
C:\DOS\SHARE.EXE /L:500
```

As you probably know, DOS automatically runs AUTOEXEC.BAT when you start up your computer. Therefore, by placing the SHARE command in the batch file, you won't ever need to worry about whether you loaded SHARE.

On the other hand, you may have opted not to let the SETUP program place the SHARE command in your AUTOEXEC.BAT file. If you don't know whether you're running SHARE, you should probably check. Open your AUTOEXEC.BAT file in the Notepad application and look for the SHARE statement. You'll find the Notepad icon in the Program Manager's Accessories group. Double-click the icon to open Notepad. Then, pull down the File menu and select the Open… command. Next, you type *C:\AUTOEXEC.BAT* in the File Name text box of the Open dialog box and click OK. When the file appears in the Notepad window, look for the line

```
C:\DOS\SHARE.EXE
```

The statement may or may not have the /L setting. If you can't find the line, you should probably add it.

After making your changes to the file, choose the Save command from the File menu. Since you've made a change to the AUTOEXEC.BAT file, you'll need to exit Windows and reboot your system before the change will take effect. From then on, whenever you open a database that's already in use in another instance of Access, Access will display the dialog box shown in Figure B.

Changing SHARE's default settings

By default, SHARE allocates 2,048 bytes of memory to keep track of up to 100 open files and manages a maximum of 20 file accesses. In most circumstances, this configuration is adequate. However, if the applications you're running will open more than 100 files or perform more than 20 simultaneous file accesses, you need to fine-tune SHARE's settings.

To do so, you use the /F:*size* setting, which specifies the number of bytes of memory used to keep track of open files, and the /L:*locks* setting, which specifies the number of simultaneous file accesses. For example, Microsoft suggests you use the /L:500 setting, which lets Access manage 500 simultaneous file accesses. Of course, Access doesn't access 500 files. However, it can access a single database file 500 times.

Windows for Workgroups doesn't need to use SHARE

If you're running Windows for Workgroups, you won't need to load SHARE. Windows for Workgroups uses a specially designed virtual device driver, VSHARE.386, to control access to shared data files while running in 386 Enhanced mode. Furthermore, VSHARE.386 dynamically adjusts its configuration based upon demand, so you don't need to fine-tune it.

Also, if your AUTOEXEC.BAT file loads SHARE, VSHARE.386 takes over for SHARE when you start Windows for Workgroups in 386 Enhanced mode. Therefore, you can save approximately 5Kb of conventional memory by removing the SHARE command from your system's AUTOEXEC.BAT file.

Note that VSHARE.386 isn't a simple replacement for DOS's SHARE. In fact, if you're running Windows for Workgroups, you'll find that you can open the same database file with separate instances of Access. But don't worry: VSHARE.386 protects your database file in a different way. Instead of insisting that only one application use a file at a time, it lets multiple instances of the application share the file as if they were distinct users. It protects the data one instance is using so the second instance can't corrupt the database file.

Notes

Although SHARE protects your data files from simultaneous access during multitasking operations in Windows 3.0 and 3.1, it can conflict with certain other applications. For example, you might encounter error messages when you run Setup programs of some applications.

Conclusion

Since you could run only one application at a time in DOS, you never had to worry about more than one application accessing a file at the same time. However, because Windows provides multitasking for both DOS and Windows applications, more than one application can simultaneously access your files. The result could be a file that contains unexpected data or even a corrupt file. In this article, we've explained how you can prevent this from happening by running SHARE. ◆

Tell us about yourself

Here at The Cobb Group, we want to tailor *Inside Microsoft Access* to meet your needs as closely as possible. We hope you'll take a few moments to fill out this short survey. Just photocopy this page, fill it out, and mail or fax it to us by November 30, 1993. Thanks for your input!

Return surveys to

Inside Microsoft Access
The Cobb Group
P.O. Box 35160
Louisville, KY 40232

Or fax the completed form to

(502) 491-8050

1. How would you rate your familiarity with Microsoft Access?

❑ Novice ❑ Experienced ❑ Expert

2. How would you rate your proficiency with Microsoft Windows and Windows applications in general?

❑ Novice ❑ Experienced ❑ Expert

3. Which of the following best describes how you use Access?

I primarily use Access interactively and don't create Access Basic modules at all.	❑
I create simple Access Basic funtions for performing custom calculations or to automate tasks.	❑
I write complete applications and distribute them to users.	❑
I am proficient with Access Basic and use it extensively in my work.	❑

4. Tell us what kinds of articles you'd like to see in future issues (with 1 indicating those you least want to see and 5 those you most want to see)

	LEAST				MOST
Short tips and Access shortcuts	1	2	3	4	5
Form design techniques	1	2	3	4	5
Query techniques	1	2	3	4	5
Report techniques	1	2	3	4	5
Tutorial articles on Access Basic functions	1	2	3	4	5
Advanced technical articles on using Access Basic for application development	1	2	3	4	5
Reviews of third-party products	1	2	3	4	5
Windows tips	1	2	3	4	5
Multiuser tips	1	2	3	4	5

Other ______________________________

5. Do you share data among Access and these programs?

Microsoft Excel	❑ Yes	❑ No
Quattro Pro	❑ Yes	❑ No
Quattro Pro for Windows	❑ Yes	❑ No
Lotus 1-2-3 for Windows	❑ Yes	❑ No

Other spreadsheets ______________________________

Word for Windows	❑ Yes	❑ No
WordPerfect for Windows	❑ Yes	❑ No
WordPerfect (DOS)	❑ Yes	❑ No
Ami Pro	❑ Yes	❑ No

Other word processors ______________________________

6. Are you running Access on a network (peer-to-peer or file-server based)?

❑ Yes ❑ No

If so, what kind? ______________________________

7. Are you connected to an SQL database server?

❑ Yes ❑ No

If so, what kind? ______________________________

Use this space to tell us what features you'd like Microsoft to include in future versions of Microsoft Access.

____
____
____
____
____
____
____
____
____

Using Access' Overstrike mode while editing cell and control entries

Editing

You probably know that most applications use the [Ins] key to toggle between Insert and Overstrike mode. While in Insert mode, the application *inserts* your keystrokes into existing text at the insertion point cursor. While in Overstrike mode, the application *replaces* existing characters with your keystrokes at the insertion point cursor.

Overstrike mode is a source of frustration for many users. In most applications, if you accidentally press the [Ins] key, the Overstrike mode indicator will appear in the status bar, but the application will continue to offer the insertion point cursor. Too often, you just keep typing, overwriting text you want to keep.

Because of this awkward characteristic, many users avoid Overstrike mode. However, you should reconsider Overstrike mode when you use Access. Access does more than simply display an indicator in a corner of the screen—it also highlights in the cell or control the character that you'll overwrite when you type the next character. Furthermore, after you type a character, Access automatically highlights the character you'll overwrite with the next keystroke. ◆

Generating the correct title for your mailing labels

Letters

I read with interest your article "Using the Mailing Label Wizard," in the May 1993 issue. However, the article didn't address a labeling problem I've encountered. I have a table of names and addresses that I want to print labels for. I want the labels to look like this example:

Mr. and Mrs. Smith
123 Any Street
Any Town, CA 95472

However, there are several single people on my mailing list, so the labels end up with entries like

Mr. and Smith

and

and Mrs. Smith

How can you build a mailing label report that prints the appropriate title?

James P. Foley
Sebastopol, California

The best way to handle this problem is by defining a query to examine each record's name data, determine the most appropriate title, and place the title in a new field. That way, you can base the mailing-label report on the query and use the Title field in the mailing-label report.

To demonstrate this technique, we'll use the table Mr And Mrs Names, shown in Figure A. In this case, you want the query to determine whether the Title 1 or the Title 2 field is blank and, from that information, construct the title.

Figure B on page 16 shows a query that builds the title. The first column includes all the fields from the Mr And Mrs Names table, and the three columns that follow cooperate to generate the title. Unfortunately, the expressions in the fields are too long to fit in a window, so we'll show them to you one at a time.

Figure A

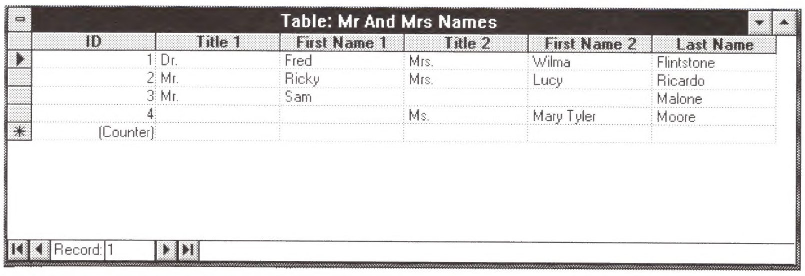


Table: Mr And Mrs Names

ID	Title 1	First Name 1	Title 2	First Name 2	Last Name
1	Dr.	Fred	Mrs.	Wilma	Flintstone
2	Mr.	Ricky	Mrs.	Lucy	Ricardo
3	Mr.	Sam			Malone
4			Ms.	Mary Tyler	Moore
(Counter)					

Record: 1

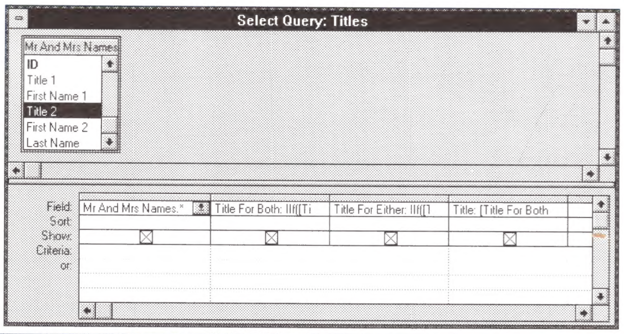
The Mr And Mrs Names table stores information for both couples and single people.

SECOND CLASS MAIL

Microsoft Access
Technical Support
(206) 635-7050

Please include account number from label with any correspondence.

Figure B



Field:	Mr And Mrs Names.*	Title For Both: IIf([Ti	Title For Either: IIf([]	Title: [Title For Both
Sort:				
Show:	☒	☒	☒	☒
Criteria:				
or:				

This query's expressions analyze the Title 1 and Title 2 fields to generate an appropriate title.

The first of these columns uses the expression

```
Title For Both: IIf([Title 1] Is Not Null And
➡ [Title 2] Is Not Null,"¦[Title 1]¦ and
➡ ¦[Title 2]¦ ",Null)
```

in the Field cell to create a field called Title For Both. This expression determines whether both Title 1 *and* Title 2 contain an entry. If so, the expression returns a string made up of the Title 1 field, the word *and*, and the Title 2 field. If either field is null, the expression returns a null value. Obviously, this field returns the title for records that store couples.

The next column uses the statement

```
Title For Either: IIf([Title For Both] Is Null
➡ And [Title 1] Is Not Null,"¦[Title 1]¦ ",
➡ Null) & IIf([Title For Both] Is Null And
➡ [Title 2] Is Not Null,
➡ "¦[Title 2]¦ ",Null)
```

in the Field cell to create the Title For Either field. This expression returns a title when either Title 1 *or* Title 2 contains a value—but not when both contain a value. As you can see, this column generates titles for single people.

The third column takes advantage of the fact that only one of the previous two columns returns a value. In other words, a record will store data for either single people or couples—not both. Its expression

```
Title: [Title For Both] & [Title For Either]
```

creates a field named simply Title, which holds the concatenation of the Title For Both and Title For Either fields. Since one field will always return a title and the other will always return a Null value, the result will be the appropriate title. ◆

Subscribe to Inside Microsoft Access Resource Disk!

Do you wish that you could experiment with the forms, reports, tables, macros, modules, and queries we regularly feature in *Inside Microsoft Access* but don't have the time or patience to create them? If so, you may want to subscribe to Inside Microsoft Access Resource Disk. Once you subscribe, we'll send you a monthly disk loaded with all the useful tips featured in that month's *Inside Microsoft Access*. See the articles marked with the disk icon.

A six-month subscription to Inside Microsoft Access Resource Disk costs $29. A full one-year subscription is $49. If you don't want to subscribe but would like the forms, reports, tables, macros, modules, and queries in a particular issue of *Inside Microsoft Access*, you can purchase a single disk for only $9.95.

To subscribe or order a specific month's disk, just call Customer Relations at (800) 223-8720. Outside the US, please call (502) 491-1900.

Printed in the USA
Printed on recycled and recyclable paper
50% recycled fiber including 10% post-consumer waste